By following a basic framework, IT managers can successfully navigate the thin line between business analyst and project manager.

Steve Gilbert

Wearing Two Hats: Analyst-Managers for Small Software Projects

As software development continues to evolve in the post dot-com era, companies have changed their project methodology and are asking IT employees to evolve with them. To this end, the pure business analyst's role has become diluted. It used to be the business analyst who talked to users, ironing out the details of what they wanted and balancing that wish list against what an IT system could economically or practically deliver. It was also the business analyst who ensured that users developed business processes to support the software. The actual task of writing the software (or installing and configuring packaged software) fell to a project manager and his team of software developers.

But those were the good old days. Today, companies commonly ask IT managers to assume the business analyst role in addition to their duties as project manager. This is especially true in small projects, those taking between 100 and 500 person-hours to complete.

Although the business analyst and project manager roles might seem quite compatible, significant conflict often occurs when a company assigns both roles to a single individual. The business analyst has traditionally served as the voice of reason during project turmoil, often stopping work to redocument or further define a particular requirement. Conversely, when faced with the same turmoil, the project manager should remain focused on the goal of delivering the project. Ideally, he must sometimes make decisions that are outside of the scope of the client's documented requests. In this article, I will use *client* to describe the business entity (either internal or external) that has requested the system or enhancement, and is ultimately responsible for funding the project. Each position thus has an inherently different focus; this difference creates unique challenges when a company asks one person to take on both roles.

STEPS FOR SMALL SOFTWARE PROJECTS

So how, as an analyst-manager, will you successfully wear two hats? Hundreds of books already cover software project management, but most concentrate on large projects that can take from six months to many years to complete. The concepts and philosophies in these books have a place in managing small projects, so I begin with a framework that arises from those sources.

Classical software development depends on the six steps outlined in Figure 1: discovery; requirements documentation; design; cost estimation and project planning; development; testing and delivery. For small projects, companies have modified these six steps, adapting them to the tight time lines and low overhead of small initiatives. These modifications are important in today's resource-pressed environment.

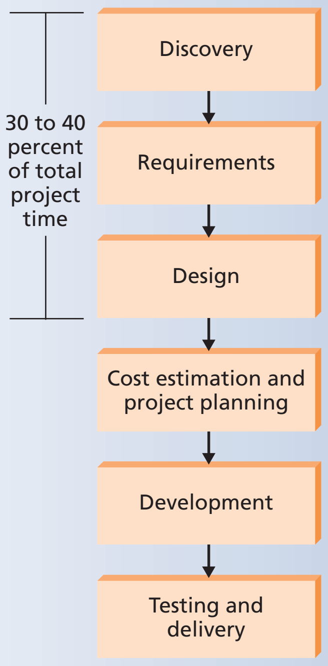
Discovery

Discovery is key to the success of any software project. Many projects in discovery are, by definition, in their infancy—just beginning and only hazily defined. At this point, it's easy to fall prey to wish list syndrome, in which users list all their desired outcomes, far beyond a project's needed focus and scope. Most initial discovery meetings include many users from throughout a company; they have their own agendas and functionality wish lists, which typically don't agree. So in this early phase, you should identify a champion: one reliable big-picture user, manager, or executive who can speak for the client. Users will then focus on that person as their primary contact throughout the project. This person should have enough clout in the client's infrastructure to have access to the resources necessary for project approval, change control, testing, and ultimately, implementation.

Because many smaller projects simply enhance or modify an existing system (or product), you must have intimate knowledge of this system and its capabilities. At this stage and with little effort, you can urge the client to use functionality or solutions now inherent in the application, or that the product's vendor has already defined and scheduled as enhancements. This knowledge will considerably increase the chances of successfully implementing the project on time and on budget. For planned small-budget projects, discovery meetings should not last more than eight hours, and within that time you and the client should accomplish the following:

- Create, solidify, and agree on a statement of the project goals; relating them back to the client's business objectives. Pay strict attention to the number of changes and their size. Project success might ultimately depend on keeping the scope and complexity of changes in check.
- Identify one person as a champion and primary contact and discuss a way for the two of you to communicate.
- Discuss the requested functionality and the client's expectations for delivery and cost.
- Sketch the major components that require modification.
- Identify major integration points and data elements. It's easy to overlook this particular component. Traps, such as the import of historical data and integration points with legacy systems, often cripple development projects at a stage too late in the game to fix.
- Discuss the change management process (for altering the project's scope), should a project need new specifications.
- Set the expectation for the project's next steps.

Figure 1. Six steps for classical software development.


30 to 40 percent of total project time
Discovery
Requirements
Design
Cost estimation and project planning
Development
Testing and delivery

Once these tasks are complete, you're ready to move on to the requirements documentation and design phases.

Requirements documentation

The requirements writing process is the single most important part of any software development project. Writing the requirements document will take at least three times longer than the discovery process. Formats for the requirements document can be as simple as a one-page, bulleted list of the major components or as elaborate as 100 pages of intense documentation containing heavily formatted text, color, and drawings. In either case, the document's contents should contain, in as much detail as possible, each and every requirement that the client has requested. You should write these requirements in simple language, avoiding technical terms that require additional clarification. Tie requirements to the appropriate business goal. Focus on the "what" and not the "how" at this stage.

For example, a requirement that focuses on a business goal would be "build automated system for accepting credit card sales via the Web." It should not yet read, "Install and configure Microsoft-SQL-Server-based database and write back-end code to transfer transaction data to merchant bank's approval system." The former is a "what" that clearly states a business need; the latter is a "how." You should perform due diligence and requirements validation *before* development starts. Failing to do so might result in costly rework later in the project.

Up to this point, this strategy might seem to follow the classic waterfall development cycle. Eventually, however, it will diverge from the classic method and incorporate other selected techniques.

When complete, a well-prepared requirements document contains the following components:

- A *revision list* includes revision numbers, dates, the author's name, and comments about the revision.
- The *introduction* contains an overview of the project and how it fits a client's business needs.
- *Scope* lists major components for development or modification.
- *Definitions* cover acronyms and terms in the context of the project. For example, it is important to expand acronyms such as XML (Extensible Markup Language) and FTP (File Transfer Protocol). Because the audience for this document is both the client and technology people, you should also specifically define business-specific terms such as "marketing lead" or "fulfillment."
- *Detailed requirements* drill down each requirement into its lowest-level business components, numbered for distinct organization. This allows for easy identification and development.
- The *assumptions* section lists any assumptions made in writing the document. An example would be, "This system is designed for use with Microsoft's Internet Explorer Version 5.5 and higher."
- *Change control methodology* is a clearly stated provision indicating that once the client and analyst-manager approve of and freeze a document, changes to any requirement must follow the predefined change control methodology.

Communicate expections for the design document before brainstorming begins.

With a document containing all these parts, you have identified client needs in a way such that the development team can understand them well enough to design and build the system. Many analyst-managers mistakenly write loose or incomplete requirements. As a result, development takes longer, and developers spend time further defining requirements instead of actually writing code.

Once the draft requirements document is complete, ask the development team (which could be one or more developers) to review the document while keeping the overall time line and estimate in mind. This discussion should take place before the document goes to the client for review. A good developer can quickly identify vague sections in requirements or outline difficult portions that the client might omit or assign to a future project.

For client delivery, it's best to save the document in a difficult-to-alter format, such as PDF or HTML. As the author, you should be responsible for changing the document's contents. Letting others review and alter the document can expose the project to additional requirements without your knowledge. A good guideline is to plan at least two revisions for each estimated 100 hours of the overall project. For example, if the project's estimated completion time is 400 person-hours, expect to change the requirement at least eight times before the client finally signs off. Only after you receive official, written authorization that the requirements document accurately lists all client expectations can the design phase begin.

Design

In small projects, it's easy to overlook the importance of the design phase. However, once you deliver the "frozen" requirements document to developers, allow them an opportunity to brainstorm and record their design. This helps to ensure that you and the developers have accounted for any large gaps between your thoughts on the project and the client's expectations. Plan for one day (eight person-hours) of design time for each 100 hours of project time and communicate expectations for the design document before brainstorming begins. A good analyst-manager attends about half of the meetings on design to answer questions about the requirements, and then allows developers the freedom to design a system on their own. Finally, developers will present their conclusions to you for consideration, review, adjustments, and approval.

Completion of the discovery, requirements, and design phases means you're 30 to 40 percent of the way through the allotted hours for the project. Although this might seem to have taken far too long, you'll discover that after writing a complete set of requirements and taking a few hours to develop a design, the actual coding does not take as long as you might expect. So, with requirements and design complete, you can now move on to estimating the remaining project hours and developing an overall project plan.

At some point during design, you can request foundational items for the project, such as the setup and configuration of a development environment or other tools. You need not wait until the entire design is complete to begin the early stages of development or environment configuration. This deviates from the classic waterfall technique, but is well within today's accepted practice.

Cost estimate and project plan

At this juncture, you can truly estimate cost and time of delivery. There is some danger at this point in the process. Typically, you will have already discussed the time line and project costs with developers. But developers will typically undershoot the actual time required. On the other hand, if the project's expectations are too aggressive, developers will argue that the team cannot deliver in the given time frame. Either way, a project manager's skills now come into play. Somewhere between managing the development team's and the client's expectations, a project time line should emerge that everyone finds acceptable. If you and the developers have trouble meeting the deadline, specific actions can put a project back on track:

- *Prioritize requirements.* Revisit the entire body of work and prioritize each individual requirement (again). This can expose those few requirements that the client actually sees as vital, and you can schedule the remaining changes for future administration or reporting releases.
- *Demonstrate a prototype.* As early as possible, demonstrate a prototype to the client's primary contact and users. Showing progress is the best way to persuade them to relax a delivery schedule. However, this technique can sometimes backfire. Showing users an interface early on might cause them to expect that the development is almost complete, while in reality you might still have a lot of work to do. Anticipate this challenge and closely manage client expectations before, during, and after the demonstration.

You should document and formally present the estimated cost and delivery date with the project time line soon after any prototype demonstration. The first reason for this formality is to secure, for contractual purposes, an official record of providing this information to the client. Second, once the project is finished, you can check for estimate accuracy, which will help improve future estimates. Generating and documenting the estimate should take no more than 8 hours for every 100 total project hours.

In many cases, providing an estimate this late in the project cycle is unrealistic. Sales people (if you work for a consultancy and serve an outside client) or your company's management (if you are working with an internal "client") typically pressure you to provide an overall project estimate early on. This often causes a mid-project correction that negatively affects the client relationship and the project's overall health. Experienced project managers avoid this trap at all costs and manage expectations at every opportunity.

Development

At this point, you've completed a majority of the analysis, and it is time to turn the project over to developers. This is where you change hats, transitioning from business analyst to project manager. Formally announce this transition with all members of the development team present. Participants at this meeting discuss the following expectations:

- *Coding practices.* The lead developer will announce his expectations in terms of coding practices, common variable names, and the use of existing system architecture. He will also specify a methodology for commenting the code.
- *Deliverables.* Depending on the project's size, you will set the expectation that the development team must show examples of progress on a daily or weekly basis. These examples include working, reliable code, provided in an environment where it functions either with the current system or with components developed by other team members. During this meeting, you will out-

Formal Theories About Software Project Management

This article contains components of various software project management philosophies. Noticeable parallels exist between the method presented here and those in the agile development, Extreme Programming (XP), and the Rational Unified Process (RUP) models.

Arthur English, a research director at Unisys Corp. Global Industries, examined the merits of XP (A. English, "Extreme Programming: It's Worth a Look," *IT Professional,* May-June 2002, pp. 48-52). English recently said he would recommend iterative development. "Whether you espouse Extreme Programming, Agile Development, or RUP, iterative development is very important," English says. "Iterations should be built into the project plan. For small projects, iterations should not go longer than a month." English added that he would recommend visual modeling using UML (Unified Modeling Language).

"In today's world of component-based development, visual modeling is essential to understanding what the requirements are and how they need to be translated into design," he said. "Requirements traceability is essential."

Rather than discounting any input or proposed theory, you should incorporate the appropriate best practices from each method into every project. Two Web sites have additional information: The Agile Alliance is a group dedicated to helping companies use agile development methods (http://www.agilealliance.com), and a Yahoo site provides helpful information about XP (http://groups.yahoo.com/group/extremeprogramming).

line the expectations for these code compatibility progress reports and give examples for what is acceptable. You will specify how frequently the team should present them.

- *Commitment*. Developers are not machines. You must lead an open discussion about vacation schedules, work environment, and hours, while setting firm productivity expectations. On a short project, you should have a daily accounting for each developer's time. Make it clear that you will not tolerate downtime and unscheduled vacations. Better yet, have the team set that expectation for its own members.
- *Overall project focus.* Explain why your company is undertaking this effort and reinforce an upbeat feeling about the project.

At the end of the meeting, leave developers to their respective project components, and encourage them to talk to the lead developer or to you if they have questions or concerns about the project requirements.

In a small project, plan for development to take between 30 and 40 percent of the hours allotted to the overall effort. In planning for small to medium deployments, account for additional hours for product-specific development, such as bug discovery or the changing of base components. These additional hours might not affect the cost estimate provided to the client, but they will change the delivery date and cost-benefit ratio for you and your development team. It is your responsibility to manage unscheduled hours closely.

To prepare for testing, you must now write test scripts. While you continue to meet with the client and developers to manage expectations, guide the work, and maintain project momentum, you are also preparing these scripts.

Testing and delivery

Software project testing has many different faces. The analyst-manager should have a formal, written test plan, using trained software testers equipped with labs of different machines using different configurations. This testing will ensure that the system is stable and ready for the client. Historically, small projects have not used an independent test team. Typically, analyst-managers did most of the testing.

Unfortunately, neither business analysts nor project managers have historically been trained or paid attention to testing procedures. This fact, combined with recently introduced practices using IT *statements of integrity*—documents that can dictate specific roles and responsibilities for project team members—have resulted in the need for a regimented and independent testing presence. Here, however, I will focus on the analyst-manager role. From a management perspective, software testing has several key components:

- *Write a test plan.* This is critical, even if the test plan includes only a short list of tests. As you become involved in an increasing number of projects, the number of tests will also grow. Remember to record known bugs in previous versions of the software, and do regression testing to be certain those same bugs are not present in the current release.
- *Test every requirement at least once.* Use the requirements document as a guide to test every client-requested function.
- *Put the software in front of users as soon as possible.* Although you've only lightly tested the software, as long as the program contains the major client-requested functions, expose it to users as early as possible for a test run. Position the demonstration as a prototype so that the client's expectations of stability are not too high. Exposing any defects early on will not extend the project's overall delivery schedule as much as finding defects later in development.

A few hours spent testing with users can save weeks of development time.

- *Produce hardcopy output when testing.* Having a hardcopy record of any error will permit better communication of any problem to developers. It also enables a comparison with future test results, as a check that developers have corrected the problem.
- *Test in the client's environment.* Simply performing tests during development or in a test lab can yield false results. Field testing is essential to the overall test effort. Visit users and provide access to the test site for use on their own workstations. Show them how to use the system and stay with them while they are testing. A few hours spent testing with users can save weeks of development later when you find, for example, a conflict with the client's environment during deployment.
- *Determine the level of testing to perform.* Involve users, using their knowledge and expectations to refine the test plan, and set expectations based on delivery and quality needs. Some have suggested that no level of testing can completely cover a product; there will always be obscure portions of code that go untested until placed in the production environment. Anomalies such as these are often acceptable to users as long as you, prior to first release, discuss and agree with them on support commitments.

The analyst-manager is responsible for determining when a product is ready for delivery, and for determining where to draw the line on diminishing returns between delivering a project on time and having a bug-free one. The management of large and small projects differs in this area. Teams on large projects build many hours into project time lines for regression and user testing. On small projects, which typically have tighter time-to-market expectations, teams might cut testing in an effort to shorten the time line, particularly for an internal client. Leaving small errors alone or permitting cryptic administrative pages might help save the project, along with time and resources.

Delivery is the final stage of most small software projects. At this point, you've documented, approved, and managed the design and development. You have tested the software and now believe it is ready for the first release. This is not a trying time for the well-organized analyst-manager, because you've already shown successive prototypes to the client over the past days or weeks. As a result, users already know what to expect from the software.

There are four major components to the delivery of a small software project:

- *Fallback position*. Give developers the option to roll back to the previous version of the software when implementing a change, should that change not react properly in the production environment.
- *Training*. While training users on the software, provide system reference materials and other resources (such as online help and technical support). These materials must explain what the software does and how to use it. Training instills users with confidence in the system, and the resulting first impression sets the tone for the entire client relationship after implementation.
- *Results confirmation*. If the new system provides output—from displaying a number on-screen, to generating a file or controlling equipment—plan to be there when the system generates its first results. Once these come in, stop the processing until you confirm that those results meet the client's expectations for accuracy and quality. Large-scale production should not proceed until after that confirmation.
- *Celebration*. After completing even a small project, acknowledge everyone on the team for their efforts, and make sure someone thanks the client for the opportunity. Members of the development team often move directly onto the project's next phase without some acknowledgment for the milestone of completing the first phase. This can breed resentment or unhappiness, and lead to attrition or a loss of productivity. You or the lead developer must be sure to acknowledge the team's accomplishment, and offer the team a short break before starting on the next project or phase.

Should it become necessary to trim the size of the development team, it is possible to have one person take on the analyst-manager role. However, I do not recommend this tactic. If the situation dictates such a dual role, integrity and a militant pursuit of communication and project structure are critical to success.

Integrity is necessary to prevent compromises in functionality or quality when scope creep or time line slippage occurs. Checks and balances, discussed by an analyst and a project manager, are a luxury of large projects. However, when one person wears both hats, a fine line exists between what is best for the client and what is best for the project.

Structured communication is mission critical to any small software project. The burden is exclusively on the analyst-manager to constantly enforce the need for documentation, structure, and control. There is little room for vague requirements, misunderstanding, or change in a small (or any) software project.

Overall, playing the analyst-manager role in a small software development project is a difficult and exhausting opportunity. One person should only undertake this endeavor in extreme circumstances where budget or time does not permit the luxury of the checks and balances that exists when a project manager and business analyst participate in the project. ■

Pertinent Books

Some of the ideas in this article originated in the following books. If you're interested in learning more about managing software development projects, consider reading these classic and still-valuable sources:

- **S. McConnell, *Rapid Development,* Microsoft Press, 1996.**
- **F.P. Brooks, *The Mythical Man Month, Essays on Software Engineering*, Addison Wesley, 1975.**

Steve Gilbert *a business systems analyst for the Boston office of Harte-Hanks Inc. of San Antonio, Tex., a worldwide, direct and targeted marketing company that provides direct marketing services and shopper advertising opportunities to a wide range of local, regional, national, and international consumer and business-to-business marketers. Contact him at steve_gilbert@harte-hanks.com.*